
WELCOME
TO
15-213
18-213
15-513

Course Overview

15-213/18-213/15-513: Introduction to Computer Systems
1st Lecture, May 22, 2018

Instructors:

Brian Railing

Sol Boucher

The course that gives CMU its "Zip"!

Overview

- **Big Picture**
  - Course theme
  - Five realities
  - How the course fits into the CS/ECE curriculum
- **Academic integrity**
- **Logistics and Policies**

The Big Picture

- **Why take this course?**
- **What do you want to learn?**

Course Theme: (Systems) Knowledge is Power!

- **Systems Knowledge**
  - How hardware (processors, memories, disk drives, network infrastructure) plus software (operating systems, compilers, libraries, network protocols) combine to support the execution of application programs
  - How you as a programmer can best use these resources
- **Useful outcomes from taking 213/513**
  - Become more effective programmers
    - Able to find and eliminate bugs efficiently
    - Able to understand and tune for program performance
  - Prepare for later “systems” classes in CS & ECE
    - Compilers, Operating Systems, Networks, Computer Architecture, Embedded Systems, Storage Systems, etc.

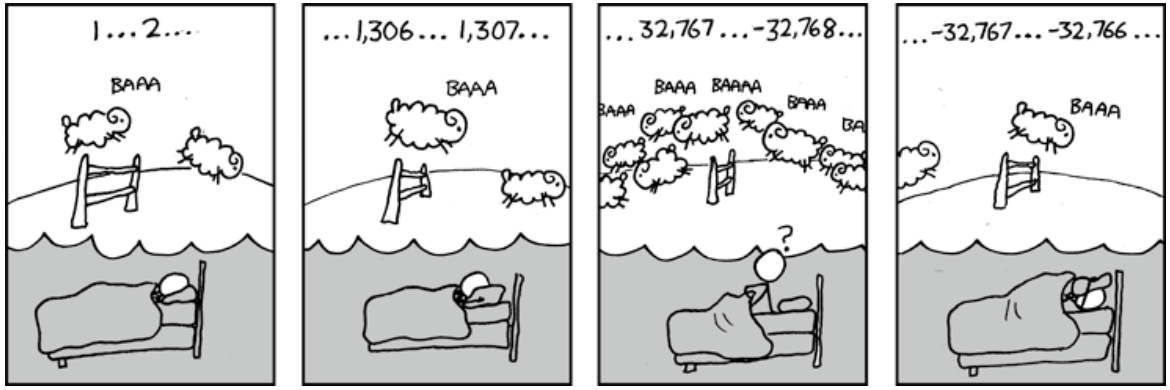
It's Important to Understand How Things Work

- **Why do I need to know this stuff?**
  - Abstraction is good, but don't forget reality
- **Most CS and CE courses emphasize abstraction**
  - Abstract data types
  - Asymptotic analysis
- **These abstractions have limits**
  - Especially in the presence of bugs
  - Need to understand details of underlying implementations
  - Sometimes the abstract interfaces don't provide the level of control or performance you need

Great Reality #1:

Ints are not Integers, Floats are not Reals

- **Example 1: Is $x^2 \geq 0$?**
  - Float's: Yes!
  - Int's:
    - 40000 * 40000 --> 1600000000
    - 50000 * 50000 --> ?



- **Example 2: Is (x + y) + z = x + (y + z)?**
  - Unsigned & Signed Int's: Yes!
  - Float's:
    - (1e20 + -1e20) + 3.14 --> 3.14
    - 1e20 + (-1e20 + 3.14) --> ??

Source: xkcd.com/571

Computer Arithmetic

- **Does not generate random values**
  - Arithmetic operations have important mathematical properties
- **Cannot assume all "usual" mathematical properties**
  - Due to finiteness of representations
  - Integer operations satisfy "ring" properties
    - Commutativity, associativity, distributivity
  - Floating point operations satisfy "ordering" properties
    - Monotonicity, values of signs
- **Observation**
  - Need to understand which abstractions apply in which contexts
  - Important issues for compiler writers and serious application programmers

Great Reality #2:
You've Got to Know Assembly

- **Chances are, you'll never write programs in assembly**
  - Compilers are much better & more patient than you are
- **But: Understanding assembly is key to machine-level execution model**
  - Behavior of programs in presence of bugs
    - High-level language models break down
  - Tuning program performance
    - Understand optimizations done / not done by the compiler
    - Understanding sources of program inefficiency
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage process state
  - Creating / fighting malware
    - x86 assembly is the language of choice!

Great Reality #3: Memory Matters

Random Access Memory Is an Unphysical Abstraction

- **Memory is not unbounded**
  - It must be allocated and managed
  - Many applications are memory dominated
- **Memory referencing bugs especially pernicious**
  - Effects are distant in both time and space
- **Memory performance is not uniform**
  - Cache and virtual memory effects can greatly affect program performance
  - Adapting program to characteristics of memory system can lead to major speed improvements

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;

double fun(int i) {
  volatile struct_t s;
  s.d = 3.14;
  s.a[i] = 1073741824; /* Possibly out of bounds */
  return s.d;
}
```

```
fun(0) -->     3.14
fun(1) -->     3.14
fun(2) -->     3.1399998664856
fun(3) -->     2.00000061035156
fun(4) -->     3.14
fun(6) -->     Segmentation fault
```

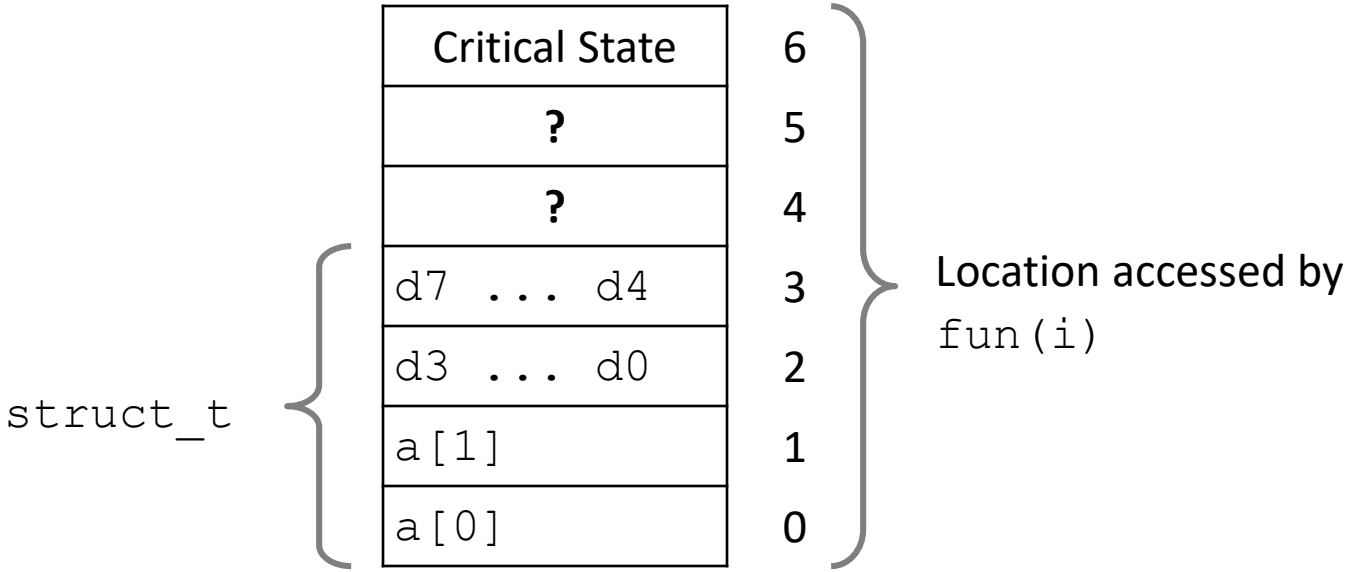
- Result is system specific

Memory Referencing Bug Example

```
typedef struct {
  int a[2];
  double d;
} struct_t;
```

```
fun(0) -->     3.14
fun(1) -->     3.14
fun(2) -->     3.1399998664856
fun(3) -->     2.0000061035156
fun(4) -->     3.14
fun(6) -->     Segmentation fault
```

Explanation:



Memory Referencing Errors

- **C and C++ do not provide any memory protection**
  - Out of bounds array references
  - Invalid pointer values
  - Abuses of malloc/free
- **Can lead to nasty bugs**
  - Whether or not bug has any effect depends on system and compiler
  - Action at a distance
    - Corrupted object logically unrelated to one being accessed
    - Effect of bug may be first observed long after it is generated
- **How can I deal with this?**
  - Program in Java, Ruby, Python, ML, …
  - Understand what possible interactions may occur
  - Use or develop tools to detect referencing errors (e.g. Valgrind)

Great Reality #4: There's more to performance than asymptotic complexity

- **Constant factors matter too!**
- **And even exact op count does not predict performance**
  - Easily see 10:1 performance range depending on how code written
  - Must optimize at multiple levels: algorithm, data representations, procedures, and loops
- **Must understand system to optimize performance**
  - How programs compiled and executed
  - How to measure program performance and identify bottlenecks
  - How to improve performance without destroying code modularity and generality

Memory System Performance Example

```
void copyij(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (i = 0; i < 2048; i++)
    for (j = 0; j < 2048; j++)
      dst[i][j] = src[i][j];
}
```

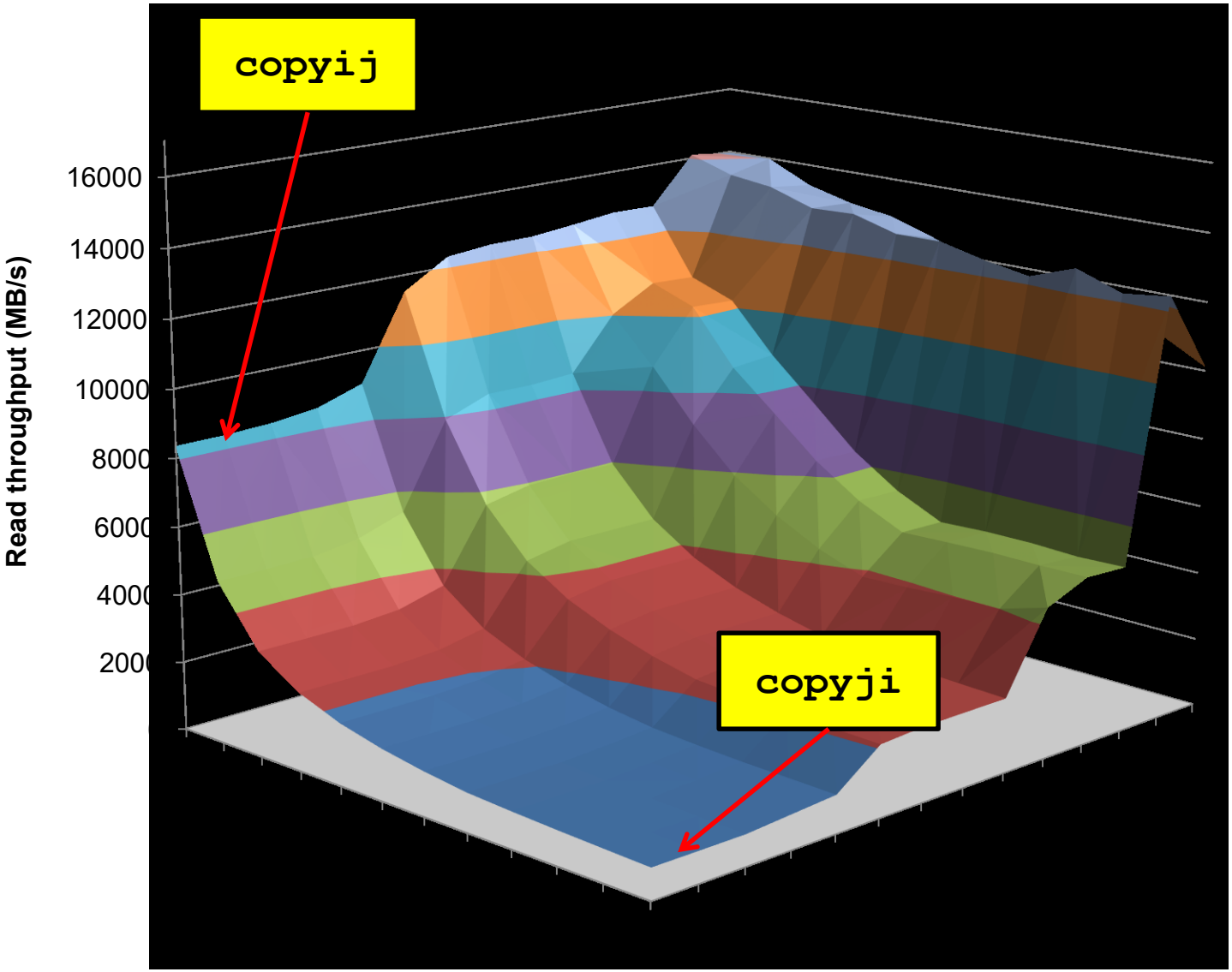
```
void copyji(int src[2048][2048],
            int dst[2048][2048])
{
  int i,j;
  for (j = 0; j < 2048; j++)
    for (i = 0; i < 2048; i++)
      dst[i][j] = src[i][j];
}
```

4.3ms **81.8ms**

2.0 GHz Intel Core i7 Haswell

- **Hierarchical memory organization**
- **Performance depends on access patterns**
  - Including how step through multi-dimensional array

Why The Performance Differs



Great Reality #5:
Computers do more than execute programs

- **They need to get data in and out**
  - I/O system critical to program reliability and performance

- **They communicate with each other over networks**
  - Many system-level issues arise in presence of network
    - Concurrent operations by autonomous processes
    - Coping with unreliable media
    - Cross platform compatibility
    - Complex performance issues

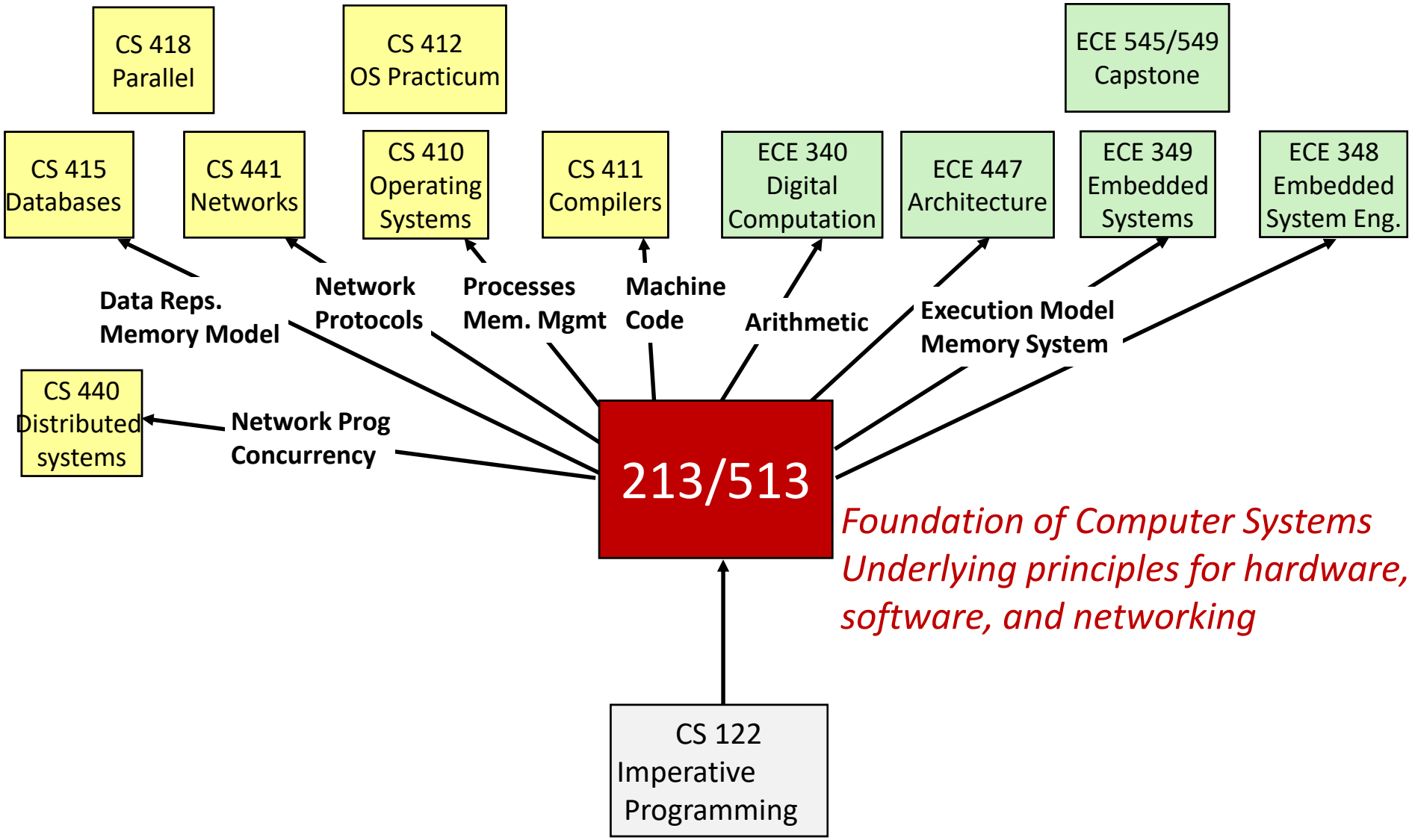
Course Perspective

- **Most Systems Courses are Builder-Centric**
  - Computer Architecture
    - Design pipelined processor in Verilog
  - Operating Systems
    - Implement sample portions of operating system
  - Compilers
    - Write compiler for simple language
  - Networking
    - Implement and simulate network protocols

Course Perspective (Cont.)

- **Our Course is Programmer-Centric**
  - By knowing more about the underlying system, you can be more effective as a programmer
  - Enable you to
    - Write programs that are more reliable and efficient
    - Incorporate features that require hooks into OS
      - E.g., concurrency, signal handlers
  - Cover material in this course that you won't see elsewhere
  - Not just a course for dedicated hackers
    - **We bring out the hidden hacker in everyone!**

Role within CS/ECE Curriculum



Foundation of Computer Systems
Underlying principles for hardware, software, and networking

Academic Integrity

Please pay close attention, especially if this is your first semester at CMU

Cheating/Plagiarism: Description

- **Unauthorized use of information**
  - Borrowing code: by copying, retyping, **looking at** a file
  - Describing: verbal description of code from one person to another.
  - Searching the Web for solutions
  - Copying code from a previous course or online solution
  - Reusing your code from a previous semester (here or elsewhere)

Cheating/Plagiarism: Description (cont.)

- **Unauthorized supplying of information**
  - Providing copy: Giving a copy of a file to someone
  - Providing access:
    - Putting material in unprotected directory
    - Putting material in unprotected code repository (e.g., Github)
  - Applies to this term and the future
    - There is no statute of limitations for academic integrity violations

Cheating/Plagiarism: Description

- **What is NOT cheating?**
  - Explaining how to use systems or tools
  - Helping others with *high-level* design issues
  - Using code supplied by us
  - Using code from the CS:APP web site

- **See the course syllabus for details.**
  - Ignorance is not an excuse

Cheating: Consequences

- **Penalty for cheating:**
  - Best case: -100% for assignment
    - You would be better off to turn in nothing
  - Worst case: Removal from course with failing grade
    - This is the default
  - Permanent mark on your record
  - Loss of respect by you, the instructors and your colleagues
  - If you do cheat – come clean asap!

- **Detection of cheating:**
  - We have sophisticated tools for detecting code plagiarism
  - In Fall 2015, 20 students were caught cheating and failed the course.
    - Some were **expelled** from the University
  - In January 2016, 11 students were penalized for cheating violations that occurred as far back as Spring 2014.

- **Don't do it!**
  - Manage your time carefully
  - Ask the staff for help when you get stuck

Some Concrete Examples:

- **This is Cheating:**
  - Searching the internet with the phrase 15-213, 15213, 213, 18213, malloclab, etc.
    - That's right, just entering it in a search engine
  - Looking at someone's code on the computer next to yours
  - Giving your code to someone else, now or in the future
  - Posting your code in a publicly accessible place on the Internet, now or in the future
  - Hacking the course infrastructure
- **This is OK (and encouraged):**
  - Googling a man page for fputs
  - Asking a friend for help with gdb
  - Asking a TA or course instructor for help, showing them your code, …
  - Looking in the textbook for a code example
  - Talking about a (high-level) approach to the lab with a classmate

How it Feels: Student and Instructor

- **Fred is desperate. He can't get his code to work and the deadline is drawing near. In panic and frustration, he searches the web and finds a solution posted by a student at U. Oklahoma on Github. He carefully strips out the comments and inserts his own. He changes the names of the variables and functions. Phew! Got it done!**
- **The course staff run checking tools that compare all submitted solutions to the solutions from this and other semesters, along with ones that are on the Web.**
  - Remember: We are as good at web searching as you are
- **Meanwhile, Fred has had an uneasy feeling: Will I get away with it? Why does my conscience bother me?**
- **Fred gets email from an instructor: "Please see me tomorrow at 9:30 am."**
  - Fred does not sleep well that night

How it Feels: Student and Instructor

- **The instructor feels frustrated. His job is to help students learn, not to be police. Every hour he spends looking at code for cheating is time that he cannot spend providing help to students. But, these cases can't be overlooked**
- **At the meeting:**
  - Instructor: "Explain why your code looks so much like the code on Github."
  - Fred: "Gee, I don't know. I guess all solutions look pretty much alike."
  - Instructor: "I don't believe you. I am going to file an academic integrity violation."
    - Fred will have the right to appeal, but the instructor does not need him to admit his guilt in order to penalize him.
- **Consequences**
  - Fred may (most likely) will be given a failing grade for the course
  - Fred will be reported to the university
  - A second AIV will lead to a disciplinary hearing
  - Fred will go through the rest of his life carrying a burden of shame
  - The instructor will experience a combination of betrayal and distress

A Scenario: Cheating or Not?

Alice is working on malloc lab and is just plain stuck. Her code is seg faulting and she doesn't know why. It is only 2 days until malloc lab is due and she has 3 other assignments due this same week. She is in the cluster.

Bob is sitting next to her. He is pretty much done.

Sitting next to Bob is Charlie. He is also stuck.

- **1. Charlie gets up for a break and Bob makes a printout of his own code and leaves it on Charlie's chair.**
  - Who cheated: Charlie? Bob?
- **2. Charlie finds the copy of Bob's malloc code, looks it over, and then copies one function, but changes the names of all the variables.**
  - Who cheated: Charlie? Bob?

Another Scenario

Alice is working on malloc lab and is just plain stuck. Her code is seg faulting and she doesn't know why. It is only 2 days until malloc lab is due and she has 3 other assignments due this same week. She is in the cluster.

Bob is sitting next to her. He is pretty much done.

Sitting next to Bob is Charlie. He is also stuck.

- **1. Bob offers to help Alice and they go over her code together.**
  - Who cheated: Bob? Alice?
- **2. Bob gets up to go to the bathroom and Charlie looks over at his screen to see how Bob implemented his free list.**
  - Who cheated: Charlie? Bob?

Another Scenario (cont.)

- **3. Alice is having trouble with GDB. She asks Bob how to set a breakpoint, and he shows her.**
  - Who cheated: Bob? Alice?
- **4. Charlie goes to a TA and asks for help**
  - Who cheated: Charlie?

- **If you are uncertain which of these constitutes cheating, and which do not, please read the syllabus carefully. If you're still uncertain, ask one of the staff**

Version Control: Your Good Friend

- All labs will be distributed via GitHub Classroom
- Must be used by all students
- Students must commit early and often
- If a student is accused of cheating (plagiarism), we will consult the GIT server and look for a reasonable commit history
- Missing GIT history will count against you
- Please make sure you have one!

Logistics

Instructors

Brian Railing

Sol Boucher

15-213/18-213 and 15-513

- **15-213/18-213**
  - Only undergraduates
  - 12 units
  - Live lectures
  - Lectures on TWR, F? 12:00-1:20 (see website)
  - Midterm 20% / Final 30%
- **15-513**
  - Only Masters students
  - 6 units
    - If you have the proper background, take 6 credits
    - If this is all new to you, take 12 credits in the Fall
  - Lectures by video (on the website and panopto)
  - Midterm 10% / Final 35%
- **Everything else is the same for all the courses**

Lecture Style

- **You are going to be active learners**
  - Come prepared to class based on the readings / videos
  - Practice and gain assessment feedback in class
  - Immediately address misconceptions with expert intervention
  - You will work in teams

- **If you have questions or concerns, please come by**
  - Or ask your advisor

Textbooks

- **Randal E. Bryant and David R. O'Hallaron,**
  - *Computer Systems: A Programmer's Perspective*, **Third Edition** (CS:APP3e), Pearson, 2016
  - http://csapp.cs.cmu.edu
  - This book really matters for the course!
    - How to solve labs
    - Practice problems typical of exam problems
  - Digital materials at: https://cmu.redshelf.com/

- **Brian Kernighan and Dennis Ritchie,**
  - *The C Programming Language*, Second Edition, Prentice Hall, 1988
  - Still the best book about C, from the originators
  - Even though it does not cover more recent extensions of C

Course Components

- **Lectures**
  - Higher level concepts
- **Recitations**
  - Material is part of lectures during summer
- **Labs (7)**
  - The heart of the course
  - 1-2+ weeks each
  - Provide in-depth understanding of an aspect of systems
  - Programming and measurement
- **Exams (midterm + final)**
  - Test your understanding of concepts & mathematical principles

Getting Help

- **Class Web page: http://www.cs.cmu.edu/~213**
  - Complete schedule of lectures, exams, and assignments
  - Copies of lectures, assignments, exams, solutions
  - FAQ
- **Piazza**
  - Best place for questions about assignments
  - By default, your posts will be private
  - We will fill the FAQ and Piazza with answers to common questions
- **Canvas**
  - Daily formative quizzes
  - Can provide access to Piazza and occasional material

Getting Help

- **Office hours (starting next week):**
  - **TAs:** TBD
  - **Faculty:** Brian Railing (GHC 6005): TBD or
    When my door is open
- **Walk-in Tutoring**
- **Staff mailing list: 15-213-staff@cs.cmu.edu**
  - Use this for logistical issues not answered on Piazza
  - Send email to individual instructors only to schedule appointments
- **1:1 Appointments**
  - You can schedule 1:1 appointments with any of the teaching staff

213 Student HowTo

- Attend Lectures
- Attend boot camps
- Start labs early (really) and use GIT properly
- TA office hours: we need to manage load and waiting time
  - lab-related concrete questions
  - must write them down before getting help
  - Time slots
- Faculty Office Hours
  - Grading, special cases, issues, lab-related questions
  - Conceptual and longer questions
  - Open discussions

Policies: Labs And Exams

- **Work groups**
  - You must work alone on all lab assignments
- **Handins**
  - Labs due at 11:59pm
  - Electronic handins using **Autolab** (no exceptions!)
- **Exams**
  - Exams will be online in network-isolated clusters
  - Held over multiple days. Self-scheduled; just sign up!
- **Appealing grades**
  - Via detailed private post to Piazza within 7 days of completion of grading
  - Follow formal procedure described in syllabus

Facilities

- **Labs will use the Intel Computer Systems Cluster**
  - The "shark machines"
  - `linux> ssh shark.ics.cs.cmu.edu`
  - 21 servers donated by Intel for 213/513
    - 10 student machines (for student logins)
    - 1 head node (for instructor logins)
    - 10 grading machines (for autograding)
  - Each server: Intel Core i7: 8 Nehalem cores, 32 GB DRAM, RHEL 6.1
  - Rack-mounted in Gates machine room
  - Login using your Andrew ID and password
- **Getting help with the cluster machines:**
  - Please direct questions to piazza

Timeliness

- **Grace days**
  - **5 grace days for the semester**
  - **Limit of 0, 1, or 2 grace days per lab used automatically**
  - Covers scheduling crunch, out-of-town trips, illnesses, minor setbacks
- **Lateness penalties**
  - Once grace day(s) used up, get penalized **15% per day**
  - No handins later than **3 days after due date**
- **Catastrophic events**
  - Major illness, deat
  - Formulate a plan (
- **Advice**
  - Once you start running late, it's really hard to catch up
  - Try to save your grace days until the last few labs

Really, Really Hard!

Other Rules of the Lecture Hall

- **Laptops: permitted**
- **Electronic communications: forbidden**
  - No email, instant messaging, cell phone calls, etc
- **Presence in lectures (213): voluntary, recommended**
- **No recordings of ANY KIND**

Policies: Grading

- **Exams (50%): midterm (20%), final (30%)**
- **Labs (50%): weighted according to effort**
- **Final grades based on a straight scale (90/80/70/60) with a small amount of curving**
  - Only upward

Programs and Data

- **Topics**
  - Bit operations, arithmetic, assembly language programs
  - Representation of C control and data structures
  - Includes aspects of architecture and compilers

- **Assignments**
  - L1 (datalab): Manipulating bits
  - L2 (bomblab): Defusing a binary bomb
  - L3 (attacklab): The basics of code injection attacks

The Memory Hierarchy

- **Topics**
  - Memory technology, memory hierarchy, caches, disks, locality
  - Includes aspects of architecture and OS

- **Assignments**
  - L4 (cachelab): Building a cache simulator and optimizing for locality.
    - Learn how to exploit locality in your programs.

Exceptional Control Flow

- **Topics**
  - Hardware exceptions, processes, process control, Unix signals, nonlocal jumps
  - Includes aspects of compilers, OS, and architecture

- **Assignments**
  - L5 (tshlab): Writing your own Unix shell.
    - A first introduction to concurrency

Virtual Memory

- **Topics**
  - Virtual memory, address translation, dynamic storage allocation
  - Includes aspects of architecture and OS

- **Assignments**
  - L6 (malloclab): Writing your own malloc package
    - Get a real feel for systems-level programming

Networking, and Concurrency

- **Topics**
  - High level and low-level I/O, network programming
  - Internet services, Web servers
  - concurrency, concurrent server design, threads
  - I/O multiplexing with select
  - Includes aspects of networking, OS, and architecture

- **Assignments**
  - L7 (proxylab): Writing your own Web proxy
    - Learn network programming and more about concurrency and synchronization.

Lab Rationale

- **Each lab has a well-defined goal such as solving a puzzle or winning a contest**
- **Doing the lab should result in new skills and concepts**
- **We try to use competition in a fun and healthy way**
  - Set a reasonable threshold for full credit
  - Post intermediate results (anonymized) on Autolab scoreboard for glory!

Doing the Lab

- **https://autolab.andrew.cmu.edu/courses/15213-m18**
  - Download the lab materials
  - (Usually as a tar file, so you will need to untar them in a new directory)
- **If you have questions**
  - Piazza
  - Office hours

Autolab (https://autolab.andrew.cmu.edu)

- **Labs are provided by the CMU Autolab system**
  - Project page: http://autolab.andrew.cmu.edu
  - Developed by CMU faculty and students
  - Key ideas: Autograding and Scoreboards
    - **Autograding:** Providing you with instant feedback.
    - **Scoreboards:** Real-time, rank-ordered, and anonymous summary.
  - Used by over 3,000 students each semester
- **With Autolab you can use your Web browser to:**
  - Download the lab materials
  - Handin your code for autograding by the Autolab server
  - View the class scoreboard
  - View the complete history of your code handins, autograded results, instructor's evaluations, and gradebook.
  - View the TA annotations of your code for Style points.

Autolab accounts

- **Students enrolled 11:45am on Tues, May 22 have Autolab accounts**
- **You must be enrolled to get an account**
  - Autolab is not tied in to the Hub's rosters
  - We will update the autolab accounts once a day, so check back in 24 hours.
- **For those who are waiting to add in, the first lab (datalab) will be available on the Schedule page of the course Web site.**

Linux/Git bootcamp

- How to tar and untar files
- How to set permissions on local and afs directories
- How to recover old files from git
- How to ssh to the lab machines
- How to use a make file
- And all the other things you were always afraid to ask …

- Watch the schedule page for date and time.

Waitlist questions

- 15-213: Amy Weis alweis@andrew.cmu.edu
- 18-213: Zara Collier (zcollier@andrew.cmu.edu)
- 15-513: Amy Weis alweis@andrew.cmu.edu

- Please don't contact the instructors with waitlist questions.

Welcome
and Enjoy!